```
#dark_median
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
from scipy import stats
import re
import pandas as pd

import os
import pathlib
import glob

os.listdir()
```

```
['frame_averages_mean.csv',
 'dark-median.ipynb',
 'dark_mean.ipynb',
 'hot_pixel_percentages.csv',
 'dark.ipynb',
 'dark',
 'msc_practical_demo_ccd_char.ipynb',
 'ccdcharacter.zip',
 'comparision.csv',
 'hot_pixel_percentages_mean.csv',
 '.ipynb_checkpoints',
 'hot_pixel_percentages_median.csv',
 'flatfield']
```

```
filenum =
glob.glob("/home/puja/Downloads/ccd_characterization/dark/temp*.fit")

filenum
```

```
['/home/puja/Downloads/ccd_characterization/dark/temp18.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp0.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp21.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp6.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp3.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp15.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp9.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp24.fit',
 '/home/puja/Downloads/ccd_characterization/dark/temp12.fit']
```

```
Gain = 2.58             # e- per ADU
sigma_threshold = 4.0   # Adjust threshold for hot pixels . Should not
be too aggresive otherwise exposure correction would be affected!
binsize = 1


# Read all files, stack, and apply hot-pixel mask
stacked_img = np.array([fits.getdata(f) - 100 for f in filenum])    #
```

```
read all files and correct for the PEDESTEL (check header)
stacked_img = stacked_img * Gain                                    #
Convert AUDs to electrons

stacked_img.shape
```

```
(9, 510, 765)
```

```
median_frame = np.median(stacked_img, axis=(1, 2), keepdims=True)  #
Per-frame median
std_frame = np.std(stacked_img, axis=(1, 2), keepdims=True)      #
Per-frame std dev


median_frame_3d = np.array([np.full(stacked_img.shape[1:], val) for
val in median_frame])
std_frame_3d = np.array([np.full(stacked_img.shape[1:], val) for val
in std_frame])



median_frame
```

```
array([[[4432.44]],

       [[3441.72]],

       [[5090.34]],

       [[3501.06]],

       [[3477.84]],

       [[3991.26]],

       [[3581.04]],

       [[6486.12]],

       [[3710.04]]])
```

```
print("Mean Frame (3D broadcasted):")
print(median_frame_3d)

print("Shape of mean_frame_3d:", median_frame_3d.shape)
```

```
Mean Frame (3D broadcasted):
[[[4432.44 4432.44 4432.44 ... 4432.44 4432.44 4432.44]
  [4432.44 4432.44 4432.44 ... 4432.44 4432.44 4432.44]
  [4432.44 4432.44 4432.44 ... 4432.44 4432.44 4432.44]
```

```
  ...
  [4432.44 4432.44 4432.44 ... 4432.44 4432.44 4432.44]
  [4432.44 4432.44 4432.44 ... 4432.44 4432.44 4432.44]
  [4432.44 4432.44 4432.44 ... 4432.44 4432.44 4432.44]]

 [[3441.72 3441.72 3441.72 ... 3441.72 3441.72 3441.72]
  [3441.72 3441.72 3441.72 ... 3441.72 3441.72 3441.72]
  [3441.72 3441.72 3441.72 ... 3441.72 3441.72 3441.72]
  ...
  [3441.72 3441.72 3441.72 ... 3441.72 3441.72 3441.72]
  [3441.72 3441.72 3441.72 ... 3441.72 3441.72 3441.72]
  [3441.72 3441.72 3441.72 ... 3441.72 3441.72 3441.72]]

 [[5090.34 5090.34 5090.34 ... 5090.34 5090.34 5090.34]
  [5090.34 5090.34 5090.34 ... 5090.34 5090.34 5090.34]
  [5090.34 5090.34 5090.34 ... 5090.34 5090.34 5090.34]
  ...
  [5090.34 5090.34 5090.34 ... 5090.34 5090.34 5090.34]
  [5090.34 5090.34 5090.34 ... 5090.34 5090.34 5090.34]
  [5090.34 5090.34 5090.34 ... 5090.34 5090.34 5090.34]]

 ...

 [[3581.04 3581.04 3581.04 ... 3581.04 3581.04 3581.04]
  [3581.04 3581.04 3581.04 ... 3581.04 3581.04 3581.04]
  [3581.04 3581.04 3581.04 ... 3581.04 3581.04 3581.04]
  ...
  [3581.04 3581.04 3581.04 ... 3581.04 3581.04 3581.04]
  [3581.04 3581.04 3581.04 ... 3581.04 3581.04 3581.04]
  [3581.04 3581.04 3581.04 ... 3581.04 3581.04 3581.04]]

 [[6486.12 6486.12 6486.12 ... 6486.12 6486.12 6486.12]
  [6486.12 6486.12 6486.12 ... 6486.12 6486.12 6486.12]
  [6486.12 6486.12 6486.12 ... 6486.12 6486.12 6486.12]
  ...
  [6486.12 6486.12 6486.12 ... 6486.12 6486.12 6486.12]
  [6486.12 6486.12 6486.12 ... 6486.12 6486.12 6486.12]
  [6486.12 6486.12 6486.12 ... 6486.12 6486.12 6486.12]]

 [[3710.04 3710.04 3710.04 ... 3710.04 3710.04 3710.04]
  [3710.04 3710.04 3710.04 ... 3710.04 3710.04 3710.04]
  [3710.04 3710.04 3710.04 ... 3710.04 3710.04 3710.04]
  ...
  [3710.04 3710.04 3710.04 ... 3710.04 3710.04 3710.04]
  [3710.04 3710.04 3710.04 ... 3710.04 3710.04 3710.04]
  [3710.04 3710.04 3710.04 ... 3710.04 3710.04 3710.04]]]
Shape of mean_frame_3d: (9, 510, 765)

std_frame
```

```
array([[[1953.41538325]],

       [[ 704.60627043]],

       [[2282.84400947]],

       [[ 988.58145467]],

       [[ 825.42297704]],

       [[1642.04981959]],

       [[1153.63382002]],

       [[2760.27943492]],

       [[1335.40190333]]])
```

```
print("std frame (3D broadcasted):")
print(std_frame_3d)


print("Shape of std_frame_3d:", std_frame_3d.shape)
```

```
std frame (3D broadcasted):
[[[1953.41538325 1953.41538325 1953.41538325 ... 1953.41538325
   1953.41538325 1953.41538325]
  [1953.41538325 1953.41538325 1953.41538325 ... 1953.41538325
   1953.41538325 1953.41538325]
  [1953.41538325 1953.41538325 1953.41538325 ... 1953.41538325
   1953.41538325 1953.41538325]
  ...
  [1953.41538325 1953.41538325 1953.41538325 ... 1953.41538325
   1953.41538325 1953.41538325]
  [1953.41538325 1953.41538325 1953.41538325 ... 1953.41538325
   1953.41538325 1953.41538325]
  [1953.41538325 1953.41538325 1953.41538325 ... 1953.41538325
   1953.41538325 1953.41538325]]

 [[ 704.60627043  704.60627043  704.60627043 ...  704.60627043
    704.60627043  704.60627043]
  [ 704.60627043  704.60627043  704.60627043 ...  704.60627043
    704.60627043  704.60627043]
  [ 704.60627043  704.60627043  704.60627043 ...  704.60627043
    704.60627043  704.60627043]
  ...
  [ 704.60627043  704.60627043  704.60627043 ...  704.60627043
    704.60627043  704.60627043]
  [ 704.60627043  704.60627043  704.60627043 ...  704.60627043
    704.60627043  704.60627043]
  [ 704.60627043  704.60627043  704.60627043 ...  704.60627043
```

```
    704.60627043  704.60627043]]

 [[2282.84400947 2282.84400947 2282.84400947 ... 2282.84400947
   2282.84400947 2282.84400947]
  [2282.84400947 2282.84400947 2282.84400947 ... 2282.84400947
   2282.84400947 2282.84400947]
  [2282.84400947 2282.84400947 2282.84400947 ... 2282.84400947
   2282.84400947 2282.84400947]
  ...
  [2282.84400947 2282.84400947 2282.84400947 ... 2282.84400947
   2282.84400947 2282.84400947]
  [2282.84400947 2282.84400947 2282.84400947 ... 2282.84400947
   2282.84400947 2282.84400947]
  [2282.84400947 2282.84400947 2282.84400947 ... 2282.84400947
   2282.84400947 2282.84400947]]

 ...

 [[1153.63382002 1153.63382002 1153.63382002 ... 1153.63382002
   1153.63382002 1153.63382002]
  [1153.63382002 1153.63382002 1153.63382002 ... 1153.63382002
   1153.63382002 1153.63382002]
  [1153.63382002 1153.63382002 1153.63382002 ... 1153.63382002
   1153.63382002 1153.63382002]
  ...
  [1153.63382002 1153.63382002 1153.63382002 ... 1153.63382002
   1153.63382002 1153.63382002]
  [1153.63382002 1153.63382002 1153.63382002 ... 1153.63382002
   1153.63382002 1153.63382002]
  [1153.63382002 1153.63382002 1153.63382002 ... 1153.63382002
   1153.63382002 1153.63382002]]

 [[2760.27943492 2760.27943492 2760.27943492 ... 2760.27943492
   2760.27943492 2760.27943492]
  [2760.27943492 2760.27943492 2760.27943492 ... 2760.27943492
   2760.27943492 2760.27943492]
  [2760.27943492 2760.27943492 2760.27943492 ... 2760.27943492
   2760.27943492 2760.27943492]
  ...
  [2760.27943492 2760.27943492 2760.27943492 ... 2760.27943492
   2760.27943492 2760.27943492]
  [2760.27943492 2760.27943492 2760.27943492 ... 2760.27943492
   2760.27943492 2760.27943492]
  [2760.27943492 2760.27943492 2760.27943492 ... 2760.27943492
   2760.27943492 2760.27943492]]

 [[1335.40190333 1335.40190333 1335.40190333 ... 1335.40190333
   1335.40190333 1335.40190333]
  [1335.40190333 1335.40190333 1335.40190333 ... 1335.40190333
   1335.40190333 1335.40190333]
```

```
  [1335.40190333 1335.40190333 1335.40190333 ... 1335.40190333
   1335.40190333 1335.40190333]
  ...
  [1335.40190333 1335.40190333 1335.40190333 ... 1335.40190333
   1335.40190333 1335.40190333]
  [1335.40190333 1335.40190333 1335.40190333 ... 1335.40190333
   1335.40190333 1335.40190333]
  [1335.40190333 1335.40190333 1335.40190333 ... 1335.40190333
   1335.40190333 1335.40190333]]]
Shape of std_frame_3d: (9, 510, 765)

# masking

hot_pixel_mask = stacked_img > (median_frame + sigma_threshold *
std_frame)  # Find hot pixels

stacked_img_filtered = np.where(hot_pixel_mask, median_frame_3d,
stacked_img)

print("Filtered Image:")
print(stacked_img_filtered)

Filtered Image:
[[[4435.02 4422.12 4406.64 ... 4447.92 4450.5  4437.6 ]
  [4680.12 4447.92 4476.3  ... 4414.38 4360.2  4422.12]
  [4489.2  4538.22 4416.96 ... 4502.1  4422.12 4411.8 ]
  ...
  [4494.36 4468.56 4481.46 ... 4416.96 4473.72 4440.18]
  [4527.9  4440.18 4481.46 ... 4471.14 4429.86 4437.6 ]
  [4435.02 4496.94 4460.82 ... 4507.26 4422.12 4406.64]]

 [[3501.06 3506.22 3464.94 ... 3436.56 3446.88 3390.12]
  [3521.7  3464.94 3501.06 ... 3441.72 3452.04 3413.34]
  [3485.58 3477.84 3472.68 ... 3426.24 3428.82 3431.4 ]
  ...
  [3506.22 3483.   3516.54 ... 3408.18 3428.82 3415.92]
  [3495.9  3483.   3477.84 ... 3408.18 3441.72 3441.72]
  [3472.68 3449.46 3467.52 ... 3444.3  3457.2  3426.24]]

 [[5175.48 4994.88 5049.06 ... 5031.   5090.34 5007.78]
  [5544.42 5180.64 5028.42 ... 5069.7  5100.66 5064.54]
  [5162.58 5077.44 5116.14 ... 5139.36 5092.92 5056.8 ]
  ...
  [5105.82 5103.24 5126.46 ... 5038.74 5098.08 5002.62]
  [5281.26 5054.22 5144.52 ... 5061.96 5105.82 4966.5 ]
  [5108.4  5020.68 5172.9  ... 5294.16 5072.28 4992.3 ]]

 ...

 [[3601.68 3612.   3601.68 ... 3550.08 3562.98 3532.02]
  [3741.   3614.58 3627.48 ... 3604.26 3534.6  3570.72]
```

```
  [3599.1  3632.64 3593.94 ... 3573.3  3532.02 3534.6 ]
  ...
  [3588.78 3593.94 3627.48 ... 3573.3  3552.66 3550.08]
  [3642.96 3609.42 3624.9  ... 3586.2  3560.4  3606.84]
  [3614.58 3632.64 3624.9  ... 3596.52 3570.72 3612.  ]]

 [[6491.28 6496.44 6540.3  ... 6501.6  6475.8  6398.4 ]
  [6955.68 6519.66 6550.62 ... 6589.32 6527.4  6437.1 ]
  [6524.82 6504.18 6444.84 ... 6643.5  6321.   6465.48]
  ...
  [6442.26 6395.82 6555.78 ... 6517.08 6496.44 6431.94]
  [6710.58 6586.74 6579.   ... 6344.22 6540.3  6442.26]
  [6501.6  6532.56 6509.34 ... 6620.28 6447.42 6370.02]]

 [[3725.52 3733.26 3715.2  ... 3653.28 3720.36 3715.2 ]
  [3846.78 3769.38 3751.32 ... 3668.76 3661.02 3686.82]
  [3725.52 3756.48 3730.68 ... 3728.1  3712.62 3686.82]
  ...
  [3702.3  3697.14 3738.42 ... 3699.72 3668.76 3730.68]
  [3820.98 3766.8  3743.58 ... 3637.8  3730.68 3699.72]
  [3725.52 3743.58 3684.24 ... 3722.94 3676.5  3710.04]]]
```

```
print("Dimensions of stacked_img:", stacked_img.shape)
print("Dimensions of stacked_img_filtered:",
stacked_img_filtered.shape)
```

```
Dimensions of stacked_img: (9, 510, 765)
Dimensions of stacked_img_filtered: (9, 510, 765)
```

```
ntot_total = stacked_img_filtered.size
```

```
nhot_total = hot_pixel_mask.sum()
```

```
ntot_total
```

```
3511350
```

```
nhot_total
```

```
7711
```

```
nhot_framewise = [hot_pixel_mask[z].sum() for z in
range(hot_pixel_mask.shape[0])]
```

```python
for z, nhot in enumerate(nhot_framewise):
    print(f"Number of hot pixels for frame z={z}: {nhot}")
```

```
Number of hot pixels for frame z=0: 926
Number of hot pixels for frame z=1: 669
Number of hot pixels for frame z=2: 961
Number of hot pixels for frame z=3: 786
Number of hot pixels for frame z=4: 741
Number of hot pixels for frame z=5: 896
Number of hot pixels for frame z=6: 845
Number of hot pixels for frame z=7: 1013
Number of hot pixels for frame z=8: 874
```

```python
ntot_framewise = [stacked_img_filtered[z].size for z in
range(stacked_img_filtered.shape[0])]


for z, ntot in enumerate(ntot_framewise):
    print(f"Total number of pixels for frame z={z}: {ntot}")
```

```
Total number of pixels for frame z=0: 390150
Total number of pixels for frame z=1: 390150
Total number of pixels for frame z=2: 390150
Total number of pixels for frame z=3: 390150
Total number of pixels for frame z=4: 390150
Total number of pixels for frame z=5: 390150
Total number of pixels for frame z=6: 390150
Total number of pixels for frame z=7: 390150
Total number of pixels for frame z=8: 390150
```

```python
for z in range(len(ntot_framewise)):

    nhot = nhot_framewise[z]
    ntot = ntot_framewise[z]


    percentage_hot_pixels = (nhot * 100) / ntot


    print(f"Frame z={z} - Number of masked pixels: {nhot}
({percentage_hot_pixels:5.3f}%)")
```

```
Frame z=0 - Number of masked pixels: 926 (0.237%)
Frame z=1 - Number of masked pixels: 669 (0.171%)
Frame z=2 - Number of masked pixels: 961 (0.246%)
Frame z=3 - Number of masked pixels: 786 (0.201%)
Frame z=4 - Number of masked pixels: 741 (0.190%)
```

```
Frame z=5 - Number of masked pixels: 896 (0.230%)
Frame z=6 - Number of masked pixels: 845 (0.217%)
Frame z=7 - Number of masked pixels: 1013 (0.260%)
Frame z=8 - Number of masked pixels: 874 (0.224%)
```

```
import csv



results = []


for z in range(len(ntot_framewise)):

    nhot = nhot_framewise[z]
    ntot = ntot_framewise[z]


    percentage_hot_pixels = (nhot * 100) / ntot


    print(f"Frame z={z} - Number of masked pixels: {nhot}
({percentage_hot_pixels:5.3f}%)")


    results.append([z, nhot, percentage_hot_pixels])


csv_file_path = "hot_pixel_percentages_median.csv"


with open(csv_file_path, mode='w', newline='') as file:
    writer = csv.writer(file)


    writer.writerows(results)

print(f"Results saved to {csv_file_path}")
```

```
Frame z=0 - Number of masked pixels: 926 (0.237%)
Frame z=1 - Number of masked pixels: 669 (0.171%)
Frame z=2 - Number of masked pixels: 961 (0.246%)
Frame z=3 - Number of masked pixels: 786 (0.201%)
Frame z=4 - Number of masked pixels: 741 (0.190%)
Frame z=5 - Number of masked pixels: 896 (0.230%)
Frame z=6 - Number of masked pixels: 845 (0.217%)
Frame z=7 - Number of masked pixels: 1013 (0.260%)
Frame z=8 - Number of masked pixels: 874 (0.224%)
Results saved to hot_pixel_percentages_median.csv
```

```
average_values = []


for z in range(stacked_img_filtered.shape[0]):

    frame_average = np.nanmean(stacked_img_filtered[z])
    average_values.append(frame_average)


average_values = np.array(average_values)


print("1D array of averages for each z frame:")
print(average_values)




average_values_df = pd.DataFrame(average_values, columns=['Average
Values'])


csv_file_path = 'frame_averages_median.csv'
average_values_df.to_csv(csv_file_path, index=False)

print(f"Averages have been saved to {csv_file_path}")
```

```
1D array of averages for each z frame:
[4458.56955755 3448.91711219 5125.38629942 3511.85277847 3484.81238139
 4011.81448274 3593.04848981 6534.67386744 3724.27425313]
Averages have been saved to frame_averages_median.csv
```

```
average_values = []


for z in range(stacked_img_filtered.shape[0]):

    frame_average = np.nanmean(stacked_img_filtered[z])
    average_values.append(frame_average)


average_values = np.array(average_values)


print("1D array of averages for each z frame:")
for avg in average_values:
    print(avg)
```

```
1D array of averages for each z frame:
4458.569557554786
3448.9171121876198
5125.3862994232995
3511.8527784698204
3484.812381391771
4011.8144827374094
3593.0484898116106
6534.673867435602
3724.27425313341
```

```
temperatures = []

for filename in filenum:
    # Use regex to extract the temperature value from the filename
(e.g., temp0.fit -> 0)
    match = re.search(r'temp(\d+)', filename)
    if match:

        temperature = int(match.group(1))
        temperatures.append(temperature)


print("Extracted temperatures from filenames:")
print(temperatures)
```

```
Extracted temperatures from filenames:
[18, 0, 21, 6, 3, 15, 9, 24, 12]
```

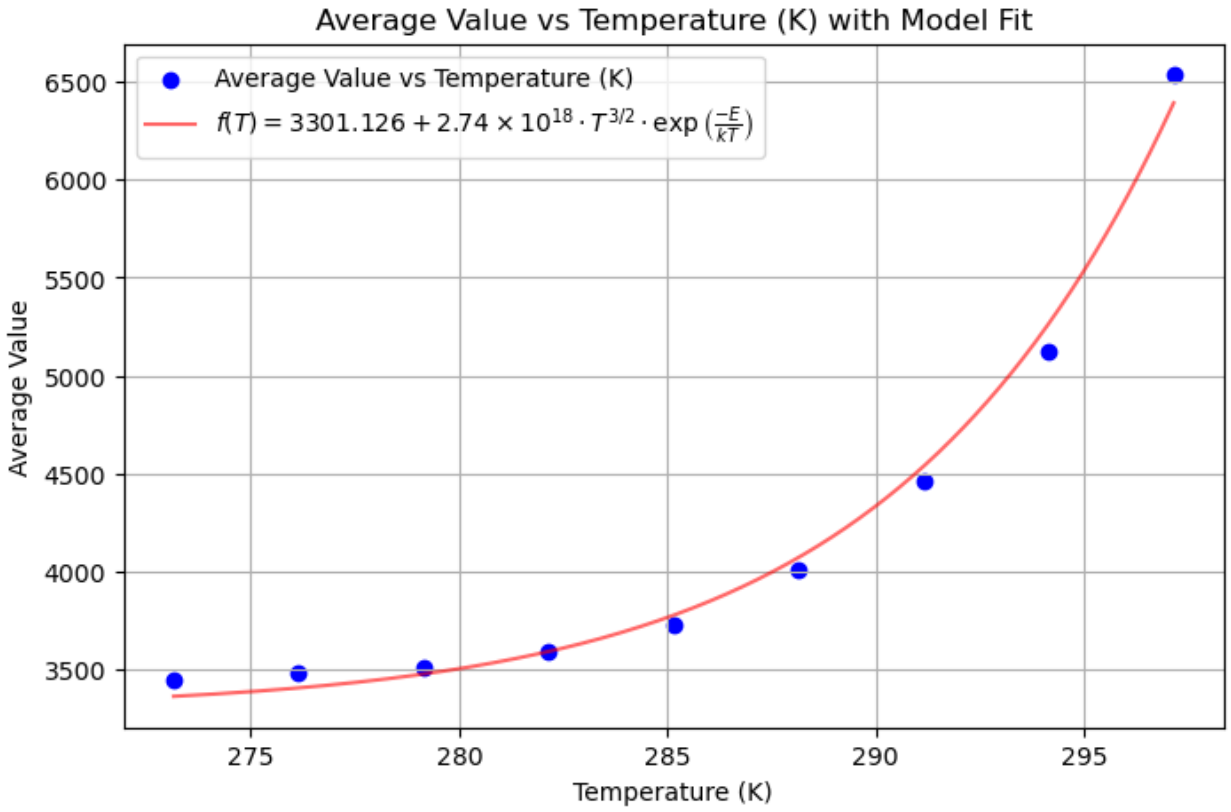
```
import matplotlib.pyplot as plt
import numpy as np


temperatures_celsius = [18, 0, 21, 6, 3, 15, 9, 24, 12]  #
temperatures in Celsius
average_values = [
    4458.942149830879, 3448.995492950635, 5125.855365199173,
    3511.9708828059993, 3484.9010040712087, 4012.0431773370624,
    3593.174322724458, 6535.29930731893, 3724.418186426005
]


temperatures_kelvin = [temp + 273.15 for temp in temperatures_celsius]


k = 8.617e-5  # Boltzmann constant in eV/K
E = 1.1  # Energy in eV
```

```python
def f_T(T, E, k):
    return T**(3/2) * np.exp(-E / (k * T))


f_values = [f_T(T, E, k) for T in temperatures_kelvin]


print("Temperature (K) | F(T)            | Average Value")
print("-------------------------------------------------")
for temp, F_T, avg in zip(temperatures_kelvin, f_values,
average_values):
    print(f"{temp:.2f}           | {F_T:.2e} | {avg:.2f}")
```

```
Temperature (K) | F(T)            | Average Value
-------------------------------------------------
291.15           | 4.51e-16 | 4458.94
273.15           | 2.28e-17 | 3449.00
294.15           | 7.17e-16 | 5125.86
279.15           | 6.44e-17 | 3511.97
276.15           | 3.85e-17 | 3484.90
288.15           | 2.82e-16 | 4012.04
282.15           | 1.06e-16 | 3593.17
297.15           | 1.13e-15 | 6535.30
285.15           | 1.74e-16 | 3724.42
```

```python
import matplotlib.pyplot as plt
import numpy as np


temperatures_celsius = [18, 0, 21, 6, 3, 15, 9, 24, 12]


temperatures_kelvin = [temp + 273.15 for temp in temperatures_celsius]


average_values = [
    4458.942149830879, 3448.995492950635, 5125.855365199173,
    3511.9708828059993, 3484.9010040712087, 4012.0431773370624,
    3593.174322724458, 6535.29930731893, 3724.418186426005
]


k = 8.617e-5  # Boltzmann constant in eV/K
E = 1.1  # Energy in eV


def f(T, E, k):
    return 3301.126 + 2.74e18 * (T**(3/2)) * np.exp(-E / (k * T))


temp_range = np.linspace(min(temperatures_kelvin),
```

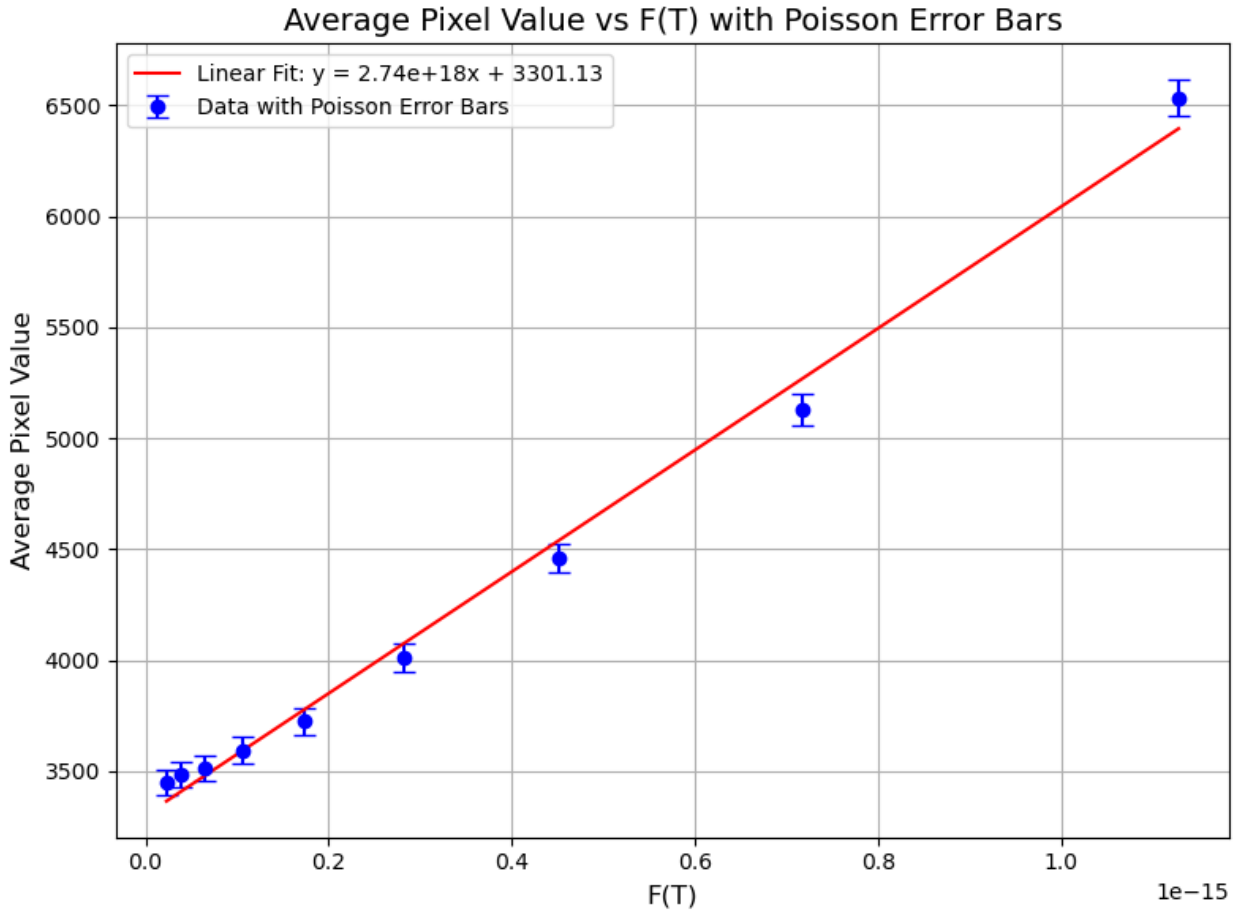
```
max(temperatures_kelvin), 1000)  # Higher resolution for smooth plot


f_values_range = f(temp_range, E, k)


plt.figure(figsize=(8, 5))


plt.scatter(temperatures_kelvin, average_values, color='b',
label='Average Value vs Temperature (K)', marker='o')


plt.plot(temp_range, f_values_range, color='r', label=r'$f(T) =
3301.126 + 2.74 \times 10^{18} \cdot T^{3/2} \cdot \exp\left(\frac{-E}
{kT}\right)$', linestyle='-', alpha=0.6)


plt.title("Average Value vs Temperature (K) with Model Fit")
plt.xlabel("Temperature (K)")
plt.ylabel("Average Value")
plt.grid(True)
plt.legend()


plt.show()
```




```python
import numpy as np
import matplotlib.pyplot as plt

# Temperatures in Kelvin
temperatures_kelvin = np.array([291.15, 273.15, 294.15, 279.15,
276.15, 288.15, 282.15, 297.15, 285.15])

# F(T) values
F_T_values = np.array([
    4.513835856256955e-16, 2.2812337037394043e-17, 7.168480485853954e-
16,
    6.435351716232285e-17, 3.8528233882041164e-17, 2.81547356106297e-
16,
    1.0634134618437857e-16, 1.1280266764201892e-15,
1.739062145655353e-16
])

# Corresponding average pixel counts
average_values = np.array([
    4458.942149830879, 3448.995492950635, 5125.855365199173,
3511.9708828059993,
    3484.9010040712087, 4012.0431773370624, 3593.174322724458,
    6535.29930731893, 3724.418186426005
])
```

```python
# Calculate Poisson errors
errors = np.sqrt(average_values)

# Sort by temperature
sorted_indices = np.argsort(temperatures_kelvin)
temperatures_sorted = temperatures_kelvin[sorted_indices]
F_T_sorted = F_T_values[sorted_indices]
average_sorted = average_values[sorted_indices]
errors_sorted = errors[sorted_indices]

# 🔢 Print sorted values
print("Temperature (K) | Average Count ± Error")
print("-" * 42)
for temp, val, err in zip(temperatures_sorted, average_sorted,
errors_sorted):
    print(f"{temp:>14.2f} | {val:.2f} ± {err:.2f}")

# 📈 Plot using original F(T) vs average values with Poisson errors
plt.figure(figsize=(8, 6))
plt.errorbar(F_T_sorted, average_sorted, yerr=errors_sorted, fmt='o',
color='blue',
             capsize=5, label="Data with Poisson Error Bars")

plt.title("Average Pixel Value vs F(T) with Poisson Error Bars",
fontsize=14)
plt.xlabel("F(T)", fontsize=12)
plt.ylabel("Average Pixel Value", fontsize=12)
plt.grid(True)

# Linear fit
fit_params = np.polyfit(F_T_sorted, average_sorted, 1)
fit_line = np.poly1d(fit_params)

# Fit line data
x_fit = np.linspace(min(F_T_sorted), max(F_T_sorted), 100)
y_fit = fit_line(x_fit)

# Plot the fit line
plt.plot(x_fit, y_fit, color='red',
         label=f"Linear Fit: y = {fit_params[0]:.2e}x +
{fit_params[1]:.2f}")

plt.legend()
plt.tight_layout()
plt.show()

# Show fit info
slope, intercept = fit_params
print(f"\nSlope: {slope:.2e}")
print(f"Intercept: {intercept:.2f}")
```

```
Temperature (K) | Average Count ± Error
----------------------------------------
        273.15 | 3449.00 ± 58.73
        276.15 | 3484.90 ± 59.03
        279.15 | 3511.97 ± 59.26
        282.15 | 3593.17 ± 59.94
        285.15 | 3724.42 ± 61.03
        288.15 | 4012.04 ± 63.34
        291.15 | 4458.94 ± 66.78
        294.15 | 5125.86 ± 71.60
        297.15 | 6535.30 ± 80.84
```




```
Slope: 2.74e+18
Intercept: 3301.13
```